DOCUMENT RESUME

ED 051 653 EM 009 004

AUTHOR	Baron, S.; And Others
TITLE	Development of Behavioral Science Programming Language. Final Report.
INSTITUTION	Bolt, Beranek and Newman, Inc., Cambridge, Mass.
SPONS AGENCY	Naval Training Device Center, Orlando, Fla.
REPORT NO	TR-NAVTRADEVCEN-70-C-0046-1
PUB DATE	Feb 71
NOTE	62p.; August 1969 - June 1970
EDRS PRICE	EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS	Behavioral Science Research, Computer Programs, *Flight Training, *Military Training, *On Line Systems, *Programing Languages, *Simulators
IDENTIFIERS	*Behavioral Science Programming Language, BSPL

ABSTRACT

Design criteria and an implementation plan are specified for a Behavioral Science Programming Language (BSPL) to be used for programing experiments on a computer-driven, high-performance aircraft simulator. The language requirements were in large part determined by the nature of the existing facility on the one hand and the types of experiments that were contemplated on the other. It was also important that the language be simple enough to be readily learned by the non-programer. This report begins, therefore, with a description of the present simulation facility, a comceptualization of a prototype experiment, and a discussion of the implication of these factors for the design of BSPL. A proposed design is then presented along with an implementation plan. The implementation plan partitions the development of the language in such a way that incremental improvements in the facility can be realized as the plan is being carried out. A formal description of the language and a sample program are included. (JY)

ED051653

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

Technical Report: NAVTRADEVCEN 70-C-0046-1

DEVELOPMENT OF BEHAVIORAL SCIENCE PROGRAMMING LANGUAGE

S. Baron
W. H. Levison
R. S. Nickerson
W. R. Sutherland
Elaine L. Thomas

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts
Contract N61339-70-C-0046
NAVTRADEVCEN Task No. 8504-1

February 1971

DoD Distribution Statement

Approved for public release; distribution is unlimited.

NAVAL TRAINING DEVICE CENTER
ORLANDO, FLORIDA

ED 009 011

NAVTRADEVCEN 70-C-0046-1

DEVELOPMENT OF BEHAVIORAL SCIENCE PROGRAMMING LANGUAGE

Abstract

Design criteria and an implementation plan are specified for a Behavioral Science Programming Language (BSPL) to be used for programming experiments on the computer-driven, high-performance aircraft simulation facility at NTDC. The BSPL was conceptualized with the following goals in mind: (a) the language should take full advantage of the existing hardware and software facilities; (b) it should be rich enough in structure to allow the programming of meaningful experiments, yet simple enough to be readily learned by the non-programmer; (c) it should be usable by individuals who do not have an intimate knowledge of the simulation program; (d) it should be incrementally implementable; and (e) it should be capable of graceful expansion and elaboration.

The language requirements are determined in large part by the nature of the existing facility on the one hand, and the types of experiments that are contemplated on the other. This report begins, therefore, with a description of the present facility, a conceptualization of a prototype experiment, and the implication of these factors for the design of the BSPL. A proposed design is then presented along with an implementation plan. The implementation plan partitions the development of the language in such a way that incremental improvements in the facility can be realized as the plan is being carried out. A formal description of the language and a sample program are included as appendices.

GOVERNMENT RIGHTS IN DATA STATEMENT

Reproduction of this publication in whole or in part is permitted for any purpose of the United States Government

AD0516553

FOREWORD

Applications of recent research results in training methods and simulation technology to the problem of training pilots and aircrews of high-performance aircraft weapon systems is one of the programs going forward at the Naval Training Device Center under Technical Development Plan N43-08X, Advanced Training Device Development.

This report describes the approach to satisfying a basic need of the behavioral scientists, who, under this program, will conduct a series of experiments on the NTDC simulation facility (TRADEC) in order to arrive at design specifications for the weapon system trainer. It is highly desirable that these scientists be able to utilize efficiently the high speed digital computer without the need for them to undergo extensive and time-consuming training in computer programming. This report contains a description of the initial efforts made in this direction. The criteria for such an enabling system and a plan for its implementation are detailed. Further efforts will include the step by step implementation of the plan as the behavioral sciences experiments are conducted.

Vincent J. Sharkey

VINCENT J. SHARKEY
Scientific Officer

TABLE OF CONTENTS

	Page
SECTION I - INTRODUCTION	1
1.1 Objective of the Study	1
1.2 Need for the BSPL	1
1.3 Approach	2
1.4 Some General Guidelines	3
SECTION II - THE PRESENT FACILITY	4
2.1 Hardware	4
2.2 Software	4
2.2.1 Real-Time Aircraft Simulation	4
2.2.2 Control and Support Function	5
2.3 Details of the Simulation Cycle	5
2.4 Limitations and Needs of the Current System	7
SECTION III - CONCEPTUALIZATION OF ROLE OF BSPL	8
3.1 Description of a Typical Experiment	8
3.1.1 Experimental Design	8
3.1.2 Implementation	8
3.1.3 Preliminary Experimentation	9
3.1.4 Formal Experimentation	9
3.1.5 Post-Experiment Analysis and Evaluation	11
3.2 Role of the BSPL in the Conduct of an Experiment.	12
3.2.1 Experimental Design	13
3.2.2 Implementation	14
3.2.3 Preliminary Experimentation	15
3.2.4 Formal Experimentation	15
3.2.5 Post-Session Analyses	16
3.3 Implications for System Improvement	16
3.3.1 Basic Programming System	16
3.3.2 New Modes of Operation	17
SECTION IV - DESCRIPTION OF THE BEHAVIORAL SCIENCE PROGRAMMING LANGUAGE	20
4.1 Off-Line Language and Translator	20
4.1.1 General Characteristics	20
4.1.2 Constructs	20

	Page
4.2 On-Line Control System	34
4.2.1 General Characteristics	34
4.2.2 Commands	34
4.2.3 Additional Features	37
4.2.4 Initialization Example	38
SECTION V - IMPLEMENTATION OF THE BSPL	39
5.1 Implementation Tasks	39
5.1.1 Basic System Modifications	39
5.1.2 Background Programs	39
5.1.3 New Simulator Modes	40
5.1.4 BSPL Translator System	40
5.2 Implementation Strategy	41
SECTION VI - REFERENCES	42
APPENDIX A - FORMAL SPECIFICATION OF THE OFF-LINE LANGUAGE	43
A.1 Command List for On-Line Control System	46
APPENDIX B - A BSPL PROGRAM TO CONDUCT AN INPUT-ADAPTIVE EXPERIMENT	47
B.1 Session Initialization	47
B.1.1 Commentary	47
B.1.2 Program Text	48
B.2 Run Initialization	49
B.2.1 Commentary	49
B.2.2 Program Text	49
B.3 Stored Experimental Module	50
B.3.1 Commentary	50
B.3.2 Program Text (with annotations)	52
B.4 Glossary of Variable Names	54

LIST OF ILLUSTRATIONS

Figure		Page
1	Proposed System Revisions	6
2	Sequence of Operations during an Experimental Session	10
3	Functional Organization of the TRADEC Simulation	18
4	Experiment Module Partitioned into States	27

SECTION I

INTRODUCTION

This report is the result of a nine-month study conducted by Bolt Beranek and Newman Inc., for the Naval Training Devices Center.

1.1 Objective of the Study

The objective of the study was to develop design criteria and an implementation plan for a Behavioral Science Programming Language (BSPL) that is to be used for programming experiments on a computer-driven, high-performance aircraft simulator.

1.2 Need for the BSPL

A computer-driven simulation facility currently exists at NTDC. It and its associated software are known collectively as the TRADEC system. The system (described below in Section II) is a sophisticated one, and represents a facility of considerable potential for experimentation in the area of simulation training. For example, facilities are provided for conditional recording of experimental variables and parameters. At present, however, any digital coding changes which might be desired must be done in a machine-like language, and for that reason the system is difficult for a nonprogrammer-experimenter to change in any significant way. The flexibility inherent in the existing TRADEC system can be exploited for a wide number of experiments, but for other experiments requiring unusual program features not already provided, machine code additions are required. If a psychologist, for example, wished to use the system to conduct a series of experiments on the relative effectiveness of a variety of simulation training procedures, he would either have to invest a large amount of time in becoming a skilled programmer himself, or to avail himself of sophisticated programming talent elsewhere. Moreover, even given a thorough understanding of the existing TRADEC software, and assuming a high degree of programming skill, implementation of each new experiment would still prove to be a considerable undertaking.

There is a need, then, for a high-level, user-oriented language to enhance the system's effective and efficient use for experimentation. We should note that during the course of this study we have been made aware repeatedly that the high-performance aircraft simulator is a complex device and that using it effectively and efficiently is never going to be a trivial task. It is unrealistic to assume that any language that might be developed could make it so. It is reasonable, however, to expect that a properly designed language could make the system much easier to use than it now is, particularly for that class of experiment requiring digital program modifications.

1.3 Approach

A basic decision that had to be made in this project was whether to start "from scratch," or to attempt to build upon what already exists. The more general approach would be to design a completely new language to perform all the simulation and data-analysis operations as well as the data-collection procedures. The alternative is to design a language to interface with the existing software.

The second and more limited approach was chosen in this case, because of the considerable amount of time that would be required to develop a completely new language. Building upon what already exists (as well as modifying existing programs where necessary) will provide NTDC with a language sufficient to meet the experimentation needs as rapidly as possible. Design of a more general language could then proceed, if desired, without causing delays in experimental programs currently under consideration. Having made this decision concerning the approach to be taken, it follows that our first major task was to obtain a thorough familiarity with the facility (particularly the software) as it currently existed. A description of the system as we found it is given in Section II.

1.4 Some General Guidelines

The following we consider to be the general principles that should guide the development of the BSPL.

—The language should take full advantage of the existing facilities, both hardware and software.

—It should be sufficiently simple in structure to be readily learned, but not so simple as to preclude the programming of non-trivial experiments.

—It should be usable by individuals who do not have a knowledge of the details of how the simulation program works.

—It should be incrementally implementable.

—It should be capable of graceful expansion and elaboration.

SECTION II

THE PRESENT FACILITY

2.1 Hardware

The simulation hardware consists of an XDS Sigma-7 digital computer, a cockpit motion system providing four degrees of freedom to a simulated F4E cockpit, a console operator's station, and assorted input/output devices. The Sigma-7 computer is a 32-bit digital machine with currently 32K of core memory, although an expansion to 48K is planned in the near future. The cockpit motion system provides cues to the pilot in roll, yaw, pitch, and heave. The console operator's station contains instruments for monitoring the aircraft and pilot performance, as well as controls for both the digital computer program and the hydraulic cockpit motion system. A complex set of analog-to-digital and digital-to-analog converters interface the cockpit instruments and cockpit motion system to the Sigma-7 computer.

2.2 Software

While the TRADEC system is in principle able to simulate virtually any aircraft, it has so far been programmed only with the equations of an F4E. The existing simulation program, which is documented in Refs. 1 and 2, runs in the bare Sigma-7 computer and consists of complex code written in assembly language. At present, no XDS computer operating system is used. A module of the program entitled "Mode and Cycle Control" serves as an executive for the system. The simulation program is divided conceptually into two main parts: (a) real-time aircraft simulation, and (b) control and support functions.

2.2.1 Real-Time Aircraft Simulation

The aircraft simulation operates in real time and is driven by a clock interrupt with a period of 50 milliseconds. It can operate in a number of modes, four of which are presently implemented: (a) normal, which simulates a flyable aircraft, (b) freeze, which suspends the updating of the aircraft equations of motion, (c) zero, which flies the aircraft from an

an arbitrary flight condition back to an initial configuration on the ground, and (d) instrument test which is used to check out the simulation hardware. The concept of modes is an important one to the simulation.

The real-time simulation functions are implemented as a series of program modules partitioned on a functional basis. The operating modes are realized by controlling which set of simulation modules are activated on each clock cycle. Thus, for example, the normal mode uses the full set of aircraft simulation modules, while the freeze mode retains the input/output operations that are necessary to keep the cockpit hardware stable while removing the aircraft variable updating modules from operation.

2.2.2 Control and Support Function

Control and support functions are performed in the background and operate essentially independently from the real-time simulation program. These background functions include such tasks as the recording of data, the printing of initial conditions of simulation variables, and the printout, after an experimental run, of data recorded during the run on magnetic tape.

2.3 Details of the Simulation Cycle

Figure 1 depicts a simplified conceptual organization of the simulation program. The 50-msec loop starts with a module which converts the raw data input from cockpit to console controls to an appropriately scaled form. The complex simulation program, shown here only as a simple module, then performs the required integration and other numerical operations to update the aircraft conditions. It also places appropriate values in the output tables. Finally, the I/O control module sends out new data to the cockpit instruments and motion system, and reads in control positions for the next cycle. After completing these real-time operations, the computer reverts to background operation. Approximately 35 milliseconds are consumed by the simulation program each 50-millisecond cycle.

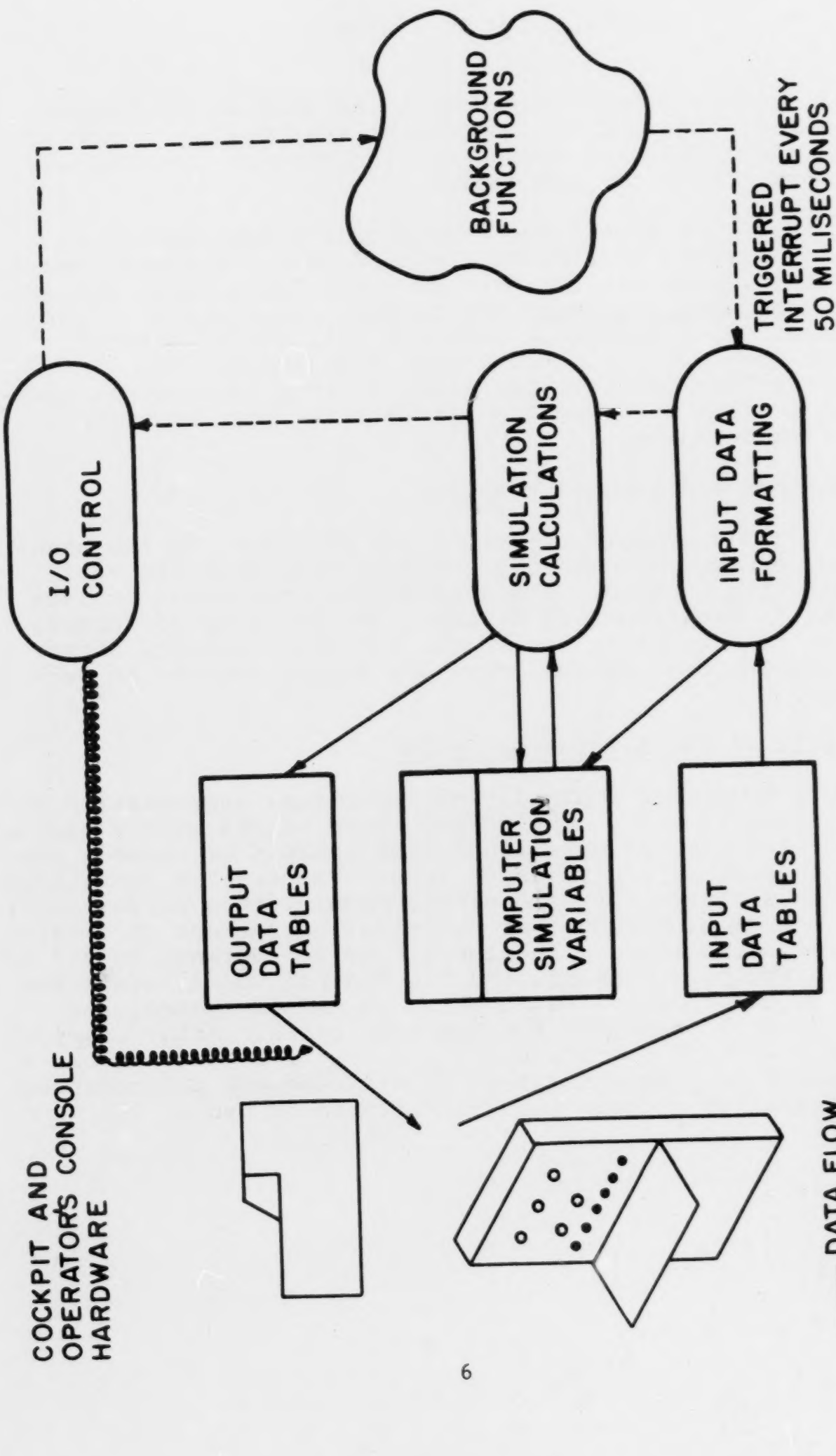

Figure 1. Functional Organization Of The TRADEC Simulation

2.4 Limitations and Needs of the Current System

The necessity to program the TRADEC system in a machine-like language renders the entire system relatively inflexible from the point of view of the non-programmer. The inability to modify the simulation cycle in a simple way limits the nature of the experiments that can be contemplated, and the difficulty of interacting with the simulation on-line makes the debugging and refining of an experiment extremely difficult.

There is, then, a need for a language that will make it feasible for the user to modify the stored simulation routines or add new ones, and for an executive system that will facilitate on-line interaction with the simulation.

SECTION III

CONCEPTUALIZATION OF ROLE OF BSPL

We have designed a Behavioral Science Programming Language (BSPL) to enable the psychologist to conduct experiments on the TRADEC system with greater ease and flexibility than is currently possible. In order to understand the way in which the BSPL will accomplish this aim, it is first necessary to understand the basic routine followed in the conduct of a typical computer-controlled experiment. Accordingly, we begin this section with a general discussion of experimental procedures. We then illustrate the role that the BSPL is expected to play at the various stages of the experimental program. Based on this discussion, we suggest the general structure of the BSPL.

3.1 Description of a Typical Experiment

We may think of the typical computer-controlled experiment as being comprised of five phases: (1) experimental design, (2) implementation, (3) preliminary experimentation, (4) formal experimentation, and (5) post-run analysis and evaluation.

3.1.1 Experimental Design

The experimenter must first define the basic objectives of his experimental program in terms of the questions he is trying to answer and the specific theories (if any) that are to be tested. Having made these basic decisions, he then proceeds to formulate the specific experimental procedure to be followed. In particular, he must decide the nature of his experimental task and the apparatus on which it is to be performed, determine the experimental variables, determine the mechanisms by which these variables should be varied as well as the appropriate ranges of variation, determine the performance measures to be taken, and fix the values of the experimental constants.

3.1.2 Implementation

Having formulated the design of the experiment, the experimenter must then decide which operations will require special

hardware, and which can be performed by existing hardware supplemented by additional digital programming. Let us assume that one is constrained to use existing facilities and that implementation of the experiment requires only a digital programming effort. The experimenter will then have to do the following: (a) write the program, (b) compile, load and debug the program, and (c) store an operating version of the program on magnetic tape.

3.1.3 Preliminary Experimentation

A certain amount of preliminary experimentation is generally needed in order to refine the details of the experiment. The values of experimental variables (and of some constants as well) are often determined on a trial-and-error basis. In addition, extensive running of the program will usually be needed to detect the various logical flaws which seem to appear in most digital programs. Flaws related to *programming* errors (such as illegal commands, misspelled variable names, and the like) will usually be detected during compilation or loading of the program. *Logical* errors (such as programming the wrong equations or improper initialization procedures) will often be detected only after the experimenter discovers that the system does not behave as expected. Repeated operation of the system may be required before all the errors are detected and corrected.

3.1.4 Formal Experimentation

A formal experimental "session" consists of a number of individual experimental "runs" usually performed in close succession. While the basic task is the same for all runs in a given session, the experimenter will change one or more of the experimental conditions from one run to the next (or from one group of runs to the next). Conditions are usually held fixed during the course of a single run.

The sequence of operations during an experimental session is diagrammed in Fig. 2. The session begins with an initialization procedure, typically performed only once, which might consist of turning on the simulation hardware, loading the digital program into the computer, and initializing those program

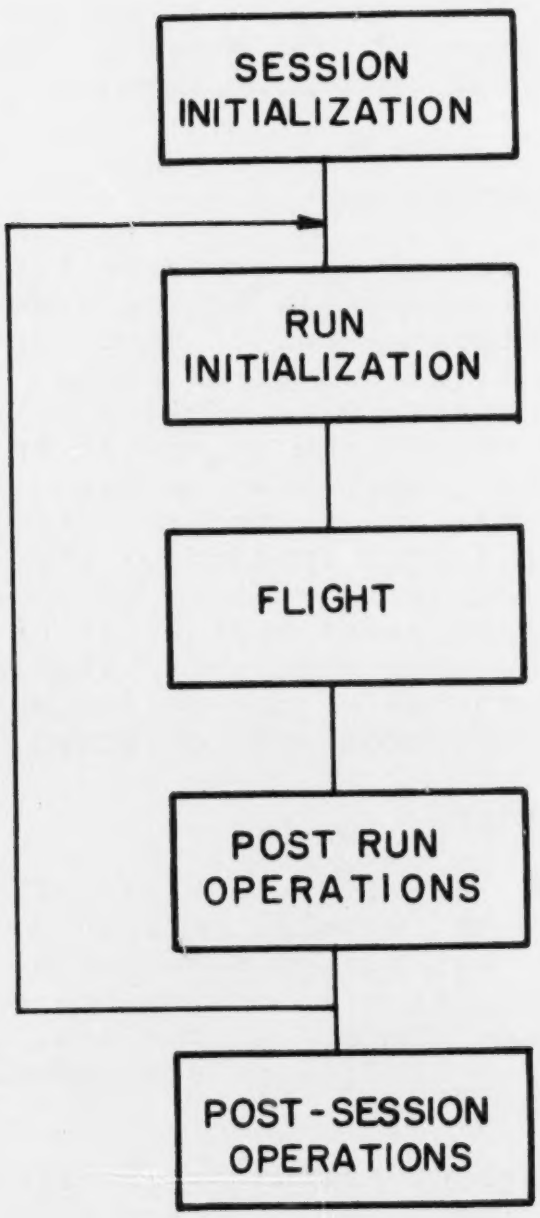

Figure 2. Sequence Of Operations During An Experimental Session

variables that are to be kept fixed for the duration of the session. The bulk of the session is devoted to the conducting of the individual experimental runs. Finally, certain post-session operations are performed after the last experimental run has been completed, the simulation equipment is turned off, and the data may then be analyzed off-line.

A single experimental run consists of three phases performed in the following sequence: (a) run initialization, (b) the "flight," and (c) immediate post-run operations. During the initialization phase the experimenter will typically initialize the simulator to the required vehicle configuration, specify certain experimental variables (such as the input level), update the run number, enter the name of the subject, and store this information at the beginning of the data tape. During the experimental flight phase the equations of motion of the vehicle are simulated, data recording and monitoring proceeds, and the contents of registers which are related to on-line data analysis are updated. ("On-line analysis" refers to the computation of various performance measures, such as root-mean-squarred error, which are computed during the course of the run so that the experimenter will have immediate knowledge of system performance at the termination of each flight.) When the flight has terminated, the results of the on-line analysis are printed, writing on the data tape is terminated, and the simulation is left in an appropriate state.

3.1.5 Post-Experiment Analysis and Evaluation

Data analysis in addition to that performed during the course of the experimental runs is required for proper evaluation of the body of experimental results. Two types of analysis procedures are often utilized: those which pertain to analysis of individual experimental runs, and those which attempt to correlate the results of many runs, or even many sessions, through appropriate statistical techniques. Finally, these results are used by the experimenter to interpret the significance of his data.

Analysis procedures which require that entire waveforms be present in core must necessarily be performed after the experimental run is completed. (Frequency-domain analyses such as signal spectra and pilot describing functions often fall into this class of procedures.) In addition, procedures which consume a significant amount of computational time cannot be performed during simulation because of the time required by the simulation itself. Although computations of this sort could be performed following each experimental run, it is usually more efficient to complete a series of runs without interruption and then analyze the data. Statistical procedures, such as an analysis of variance, must typically be deferred until the entire body of experimental data has been obtained.

3.2 Role of the BSPL in the Conduct of an Experiment

The role that the BSPL might play in each of the experimental phases is best illustrated in terms of a specific experiment. Accordingly, we shall make repeated reference to the input-adaptive experiment of Lowes, et al. (Ref. 3). We choose this experiment as our "prototype" because it is familiar to the personnel of NTDC, it is representative of the kinds of experiments we expect to be performed on the NTDC facility in the near future, and it provides a good vehicle for illustrating the BSPL language.*

The BSPL should play a direct role in the implementation, preliminary experimentation, and formal experimentation phases of the experiment. In addition, it is bound to have an important indirect role in the design and evaluation phases to the extent that the nature of the experiment is influenced by its capabilities and limitations.

---

*Appendix B contains a sample BSPL program to conduct, on the NTDC facility, an experiment of the type described in Ref. 3.

3.2.1 Experimental Design

Let us assume that the experimenter wishes to perform an experiment on the TRADEC system to test the efficiency of a self-adaptive training technique. If he were to repeat the experiment of Ref. 3, he would assign the pilot the task of maintaining a constant altitude in the presence of wind-gust disturbances. The primary experimental variable would be the "adaptiveness" of the rough air amplitude: that is, amplitude would either be determined continuously according to the pilot's instantaneous performance (the "experimental," or self-adaptive condition), or else it would be fixed at a level determined by the experimenter prior to the run (the "control" condition). An additional experimental variable would be the input level specified for the "control" trials. A certain amount of preliminary experimentation might be required to enable the experimenter to decide on the particular input levels to be used.

The particular way in which the input amplitude is to be varied must be specified in an appropriate set of equations. For the example in Ref. 3, the input amplitude was varied adaptively with respect to the pilot's ability to hold a constant altitude according to the following set of equations:

$$\Delta G = +k(h_c - |h_a - h_r|)^2 \quad \text{if } h_c - |h_a - h_r| > 0$$

$$\Delta G = -k(h_c - |h_a - h_r|)^2 \quad \text{if } h_c - |h_a - h_r| < 0 \qquad (1)$$

where ΔG is the change in the "input gain" which acts as a miltiplier on the wind-gust disturbance, h_c is a criterion altitude error, h_a is the actual aircraft altitude, and h_r is the reference or desired altitude. Input gain was thus manipulated in a continuous manner so that the short-term

average absolute error in altitude was maintained at a prespecified level. The adaptive rate constant k was chosen to provide the proper sensitivity of input gain to altitude error.

3.2.2 Implementation

Because of the necessity to encode in a machine-like language, modification of the current TRADEC programs to accommodate the experiment described above would be a relatively complicated task. The BSPL, however, would allow the experimenter to modify the simulation code so that the rough air gain would be adjusted appropriately each simulation cycle. In addition, the user-generated portion of the program could be expanded to include computation of root-mean-squared errors and histograms, data recording on magtape, and data monitoring via the strip-chart recorder.

Additional programming capability would relieve some of the burden on the experimenter's memory during the course of the experiment and would thus reduce the probability of human error. For example, certain experimental variables are usually specified at the beginning of each run. (In the case of an input-adaptive experiment, one might specify the initial value of input gain and, in addition, set a Boolean variable equal to 1 or Ø according to whether or not the input is to be self-adaptive.) A stored routine which requests numerical values for these parameters at the beginning of each run would prevent the experimenter from forgetting to specify one or both of the variables. Similarly, a routine can be written and stored which will assure that the simulator is initialized properly before any data are recorded or analyzed.

When the experimental additions to the basic system had been programmed, they would be translated into machine-language code which would then be loaded into the digital computer with the basic TRADEC simulation code. The result would be a specific executable program containing the basic simulation code and the special program components unique to the experiment. This code might then be stored on magnetic tape which could be used at a later time to restore the state of the computer and establish the appropriate experimental environment.

In general, a new program would have to be written, compiled, loaded, and stored when a new experiment is designed or when major changes are made to an existing experiment. If the experimenter is supplied with a sufficiently powerful language, however, he should be able to write his program in such a way that it will accommodate minor changes in the experiment without recompilation.

3.2.3 Preliminary Experimentation

We have pointed out that a certain amount of preliminary experimentation is needed in general to enable the experimenter to choose certain experimental variables and constants. We expect that this would be very much the case for the input-adaptive experiment. For example, the experimenter may wish to choose a single value for the adaptive rate constant that is appropriate for all experimental conditions. This constant should be numerically large enough so that the adaptive variable is reasonably responsive to system performance, yet not so large that the adaptive algorithm becomes unstable. A considerable amount of experimentation may be necessary before an acceptable numerical value can be specified.

This type of trial and error "diddling" would be greatly facilitated by a suitably-designed on-line operating system. Such a system that would allow variables to be changed easily at run time would be very valuable in debugging both the compiler program and the experimental procedure.

3.2.4 Formal Experimentation

The implementation of several new modes of operation would allow the formal experiments to proceed more efficiently than at present. They are most needed to automate the initialization and termination phases of the experimental run. The new modes that we suggest are described in Section 3.3.2.

The flight phase of the experimental program will generally proceed without operator intervention for a length of time that is either pre-specified or computed during the flight itself. Nevertheless, the operator should have the facility to terminate

the flight at any time he considers appropriate, and either place the simulation in a "hold" or "freeze" condition or restore it to some desired initial condition. In addition, the operator should be able to modify the numerical value of any system parameter without interrupting the flight. This capability would be particularly useful during the preliminary experimental and debugging phase of the program.

3.2.5 Post-Session Analyses

Since post-session analyses are, by definition, performed after the simulation (and perhaps on a different digital computer), the BSPL need not be designed to perform this function. On the contrary, satisfactory results may be most rapidly obtained if the BSPL is developed initially as a special-purpose language designed solely to facilitate the gathering of experimental data on the TRADEC system. Of-line analyses can be performed using one of the languages (such as FORTRAN-IV) that are currently available on the XDS Sigma-7. The magtape data files that are created during the experimental run will serve as the interface between the BSPL and the off-line analysis procedures. Accordingly, the BSPL must be designed so that the data are recorded in a format that is compatible with the structure of the language to be used for data analyses.

3.3 Implications for System Improvement

The combination of the present facilities and the projected experimental use of those facilities has several implications that should guide any attempts at improvement. We consider these briefly below.

3.3.1 Basic Programming System

The present simulation system may be viewed as a real-time subroutine hosted within an executive system. The real-time subroutine performs the actual simulation of the F4E aircraft characteristics, while the executive portion handles details of

computer system operation. Significant improvements in system operation from the user's point of view can be realized in both of these components. First, the F4E real-time simulation subroutine should be kept essentially intact, but methods should be provided for creating additions to it for the purpose of modifying its behavior. Therefore, we suggest the development of a language and translator designed specifically to facilitate the creation of additions to the existing simulation code. Secondly, a more flexible executive system host for the simulator is also needed. We therefore recommend the development of an interactive on-line control system which will interface the experimenter to the simulation system in a convenient and comfortable fashion.

Figure 3 shows the conceptual additions that we are recommending. The real-time simulation loop can be augmented with an experiment module and associated with additional simulation variables. This module would be created prior to any experimental runs and would in general be different for each experiment. With this module, the performance of the simulation equations could be changed, special experimental variables could be calculated and recorded, and subject performance monitored.

The on-line control system shown in Fig. 3 should replace the background functions of the original system and should be designed to provide a more flexible run-time interface to the experimenter.

3.3.2 New Modes of Operation

In addition to these rather fundamental changes, several new modes of simulator operation should be established. While the existing four modes of normal, freeze, zero, and test should be retained, additional modes are required to perform experiments and to provide a convenient capability for saving and initializing the state of the aircraft being simulated.

Specifically, we propose a new mode called "experiment" which is in all respects like "normal fly" except that the additional experimental module is incorporated in the 50-millisecond simulation loop. This experimental module would be created by the experimenter as a unique addition for each experiment. New modes of operation which we will call "dump" and "initialize" are

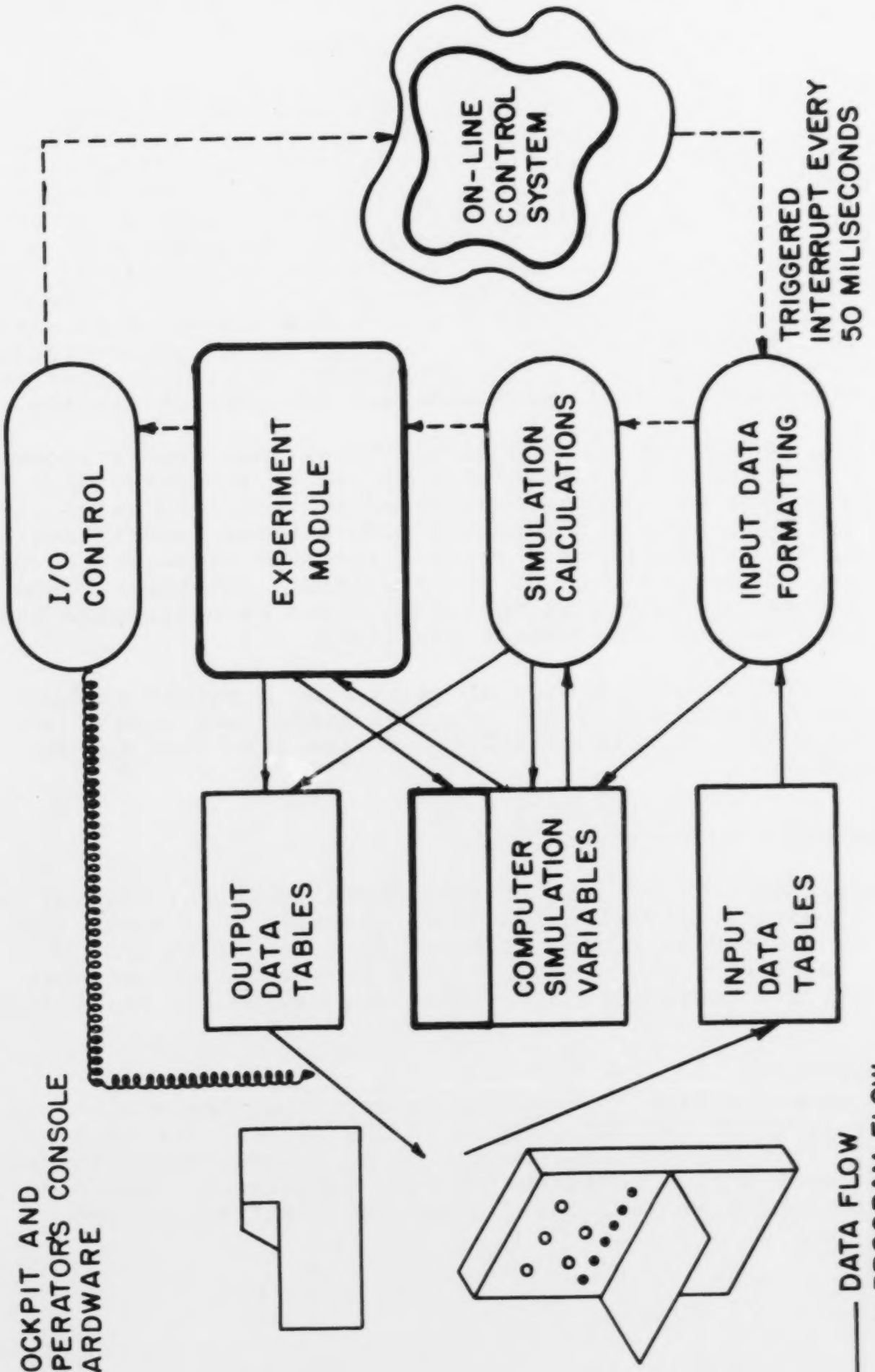

Figure 3. Proposed System Revision

desired to provide for the saving on magnetic tape of an aircraft state and for the later restoration of the simulation variables and cockpit state to the values which have been saved. Implementation of these two modes would provide a more useful capability for saving and initializing the state of the aircraft being simulated; and the necessity of manually flying the aircraft to the same initial conditions prior to each experimental run would be thereby avoided. Finally, we suggest a mode called "quit" to automate those actions required to terminate an experimental run, such as placing the simulation in the "hold" or "ground" condition, writing and end-of-file on the data tape, and rewinding the data tape.

SECTION IV

DESCRIPTION OF THE BEHAVIORAL SCIENCE PROGRAMMING LANGUAGE

In keeping with the results of the analysis of current facilities and desired improvements, we have conceptualized a Behavioral Science Programming Language (BSPL) that involves two programming subsystems: an off-line language and translator to facilitate creation of stored experimental modules which will modify the basic simulation cycle, and an on-line control system to permit control of the experiment. A formal specification for the BSPL is given in Appendix A.

4.1 Off-Line Language and Translator

4.1.1 General Characteristics

The BSPL translator operating off-line in advance of any experimental runs provides a method for creating additions to the basic simulator system for each experiment.* The constructs of the off-line language must serve both the relatively inexperienced user and the more sophisticated experimenter. To meet this need, the language has been designed with a number of high-level components which can compactly express useful experiment concepts. It also contains a simple but general programming capability to provide flexibility where needed.

4.1.2. Constructs

In a conventional programming sense, the language has constructs for declaring variable names, for specifying computer actions, and for declaring groups of actions as procedures. The basic form of a BSPL program is:

---

*For convenience, we have written this section in the present tense. The readers should understand that we are describing a proposed programming language which is not currently implemented on the TRADEC system.

```
START
    variable declarations
    special structure declarations
    procedure declarations
    action statements
FINISH
```

4.1.2.1 *Variables*

A variable is a construct that has a name and a value; the value may differ from time to time. The experimental module created by the BSPL has access to any of the "system variables" that are currently defined in the TRADEC system. (These variables are listed in Appendix A of Ref. 2.) The numerical values of system variables can be read and, if appropriate, modified by the action of the experimental module.

In addition, the user may declare new variables. Specific declaration of these variables is required as an aid to error control. Since experimental additions to the simulator program will, in general, be hard to debug, error prevention is worth some programming inconvenience.

Three kinds of variables are permitted: NUMBER, BOOLEAN, and TEXT. Declaration statements have the form:

$$\left.\begin{array}{l}\texttt{NUMBER}\\ \texttt{BOOLEAN}\\ \texttt{TEXT}\end{array}\right\} \text{name}_1, \text{name}_2, \text{name}_3, \ldots \text{name}_k;$$

Number variables are in Sigma-7 floating point format, boolean in integer form with TRUE=1 and FALSE = Ø. Text variables contain character strings.

4.1.2.2 *Operators*

An operator is a symbol that identifies an operation that is to be performed on one or more variables or constants.

Arithmetic. Conventional arithmetic operators are allowed in the usual programming sense. They include:

+ - * / ABSV SQRT

which indicate the operations of addition, subtraction, multiplication, division, absolute value, and square root, respectively.

Boolean. Conventional Boolean operators (AND, OR, NOT) are permitted.

Relational. The relational operators include:

= -= >= <= > <

The relational format 2000 <ALT<5000 is allowed.

4.1.2.3 *Expressions*

An expression is a combination of variables or constants and operators which can be evaluated to produce a numerical or boolean result.

Any number of operations may be compounded to form a single boolean expression. For example, the expression

2000<ALT<5000 AND (-25<ROLL<-20 OR 20<ROLL<25)

has the value TRUE if aircraft roll is within either of the limits indicated and the altitude is simultaneously within its limits; the value of the expression is FALSE otherwise.

4.1.2.4 *Statements*

Statements are normally executed every simulation cycle. A statement or group of statements may begin with a conditional

modifier if the action is to occur only under certain conditions or at certain times. A collection of statements may be together as a procedure. The following classes of statements are available in the experimental module.

Assignment. Assignment statements in this language are of the form

```
SET VAR1=ALT+5000;
```

This command causes the numerical value of VAR1 to be set equal to the sum of the numerical value of ALT and the number 5000.

A single command may have multiple objects. For example, the statement

```
SET VAR1=ALT+5000, VAR2=MACH/SPEED;
```

is legal.

Output. One necessary part of each experiment is the production of data in some usable form. The data output statements of the language are:

```
RECORD
LPRINT
TYPE      } variable, variable, . . .variable;
PUNCH
```

These statements produce text which is then dispatched to a background program for outputting to the mag tape, line printer, teletype, and card punch, respectively. The text is buffered to ensure that output is not sent too fast for the slower devices; some programming care is required to avoid overflowing these buffers.

Mode Control. The mode of the simulator operation can be controlled from within the simulation loop by executing one of the following statements. Each has the effect of putting the system into that mode for the following cycle.

```
FLY;          (aircraft flying)
HOLD;         (flying suspended)
GROUND;       (aircraft forced to ground)
DUMP;         (record aircraft condition)
INITIALIZE;   (restore aircraft conditions)
QUIT;         (terminate experiment)
```

The command EXPERIMENT is omitted from this list, since the experimental module is activated only when the system is in the EXPERIMENT mode.

4.1.2.5. *Modifiers*

A modifier may be used at the beginning of a statement to specify at what times or under what conditions the statement is to be executed.

Timing Control. The experimenter may wish certain actions to be performed at regular intervals but less frequently than every cycle. For example, if signal bandwidth is relatively low he may wish to record data only every fourth cycle in order to minimize the bulk of data that has to be stored and manipulated. Operations of this sort may be indicated by a special modifier having the form:

```
EVERY n statement;
```

where "n" is an integer. For example,

```
EVERY 4 RECORD ALT, MACH;
```

Recording will then take place only on every fourth simulation cycle. The form

```
ONCE statement;
```

will execute the statement only on the first execution cycle of a state. (See section on state control.) Statements without an EVERY n or ONCE prefix are executed each simulation cycle.

Conditionals. If a command is to be executed only when certain conditions are met, the statement is of the form:

```
IF boolean expression THEN statement;
```

For example,

```
IF ALT>10000 THEN SET TEMP1=ALT+5000;
```

The replacement takes place only when the boolean expression is TRUE, otherwise the value of TEMP1 is left unchanged. If some other action is to be taken when the boolean expression is FALSE, the appropriate form of the statement is:

```
IF boolean expression THEN statement ELSE statement;
```

4.1.2.6 *Compound Statements*

Any group of statements can be bracketed and used as a single compound statement. For example,

```
EVERY 10 IF -30<ROLL<30 THEN

     <RECORD VAR1, ALT, PITCH;
     PUNCH PITCH;
     SET ALT=ALT-100;>
```

Here the execution of the bracketed statements is determined by the timing and roll limit conditions preceding the THEN.

4.1.2.7 *Program States*

During the course of an experimental run, it may be desirable to obtain several distinct functional actions from the stored experimental module. For example, an individual run might be divided into (a) a monitoring phase where the subject's performance is measured as a reference, (b) a problem phase where some kind of task performance is expected and recorded, and (c) a final monitoring phase to measure and determine possible subject improvements. To accommodate experiments of this kind, it is possible to partition the experimental module into separate parts called "states," only one of which is in the simulation loop at any time. Figure 4 schematically depicts this division along with the "switch" controlling which state is active. Statements within a state can throw the switch to another state for the next cycle.

The partitioning construct is of the form

```
STATE name;
statement;
statement;
. . . .
STATE name;
statement;
. . . .
. . . .
```

Once a program state has been entered, the associated block of code is executed each simulation cycle until another state name is specified or until the simulation ceases to be in the EXPERIMENT mode. The statement

```
NEXTSTATE name;
```

will cause the named state to be entered on the following simulation cycle.

The following examples illustrate some of the constructs explained so far and also demonstrates the advantage of partitioning the program into states. Consider the following program:

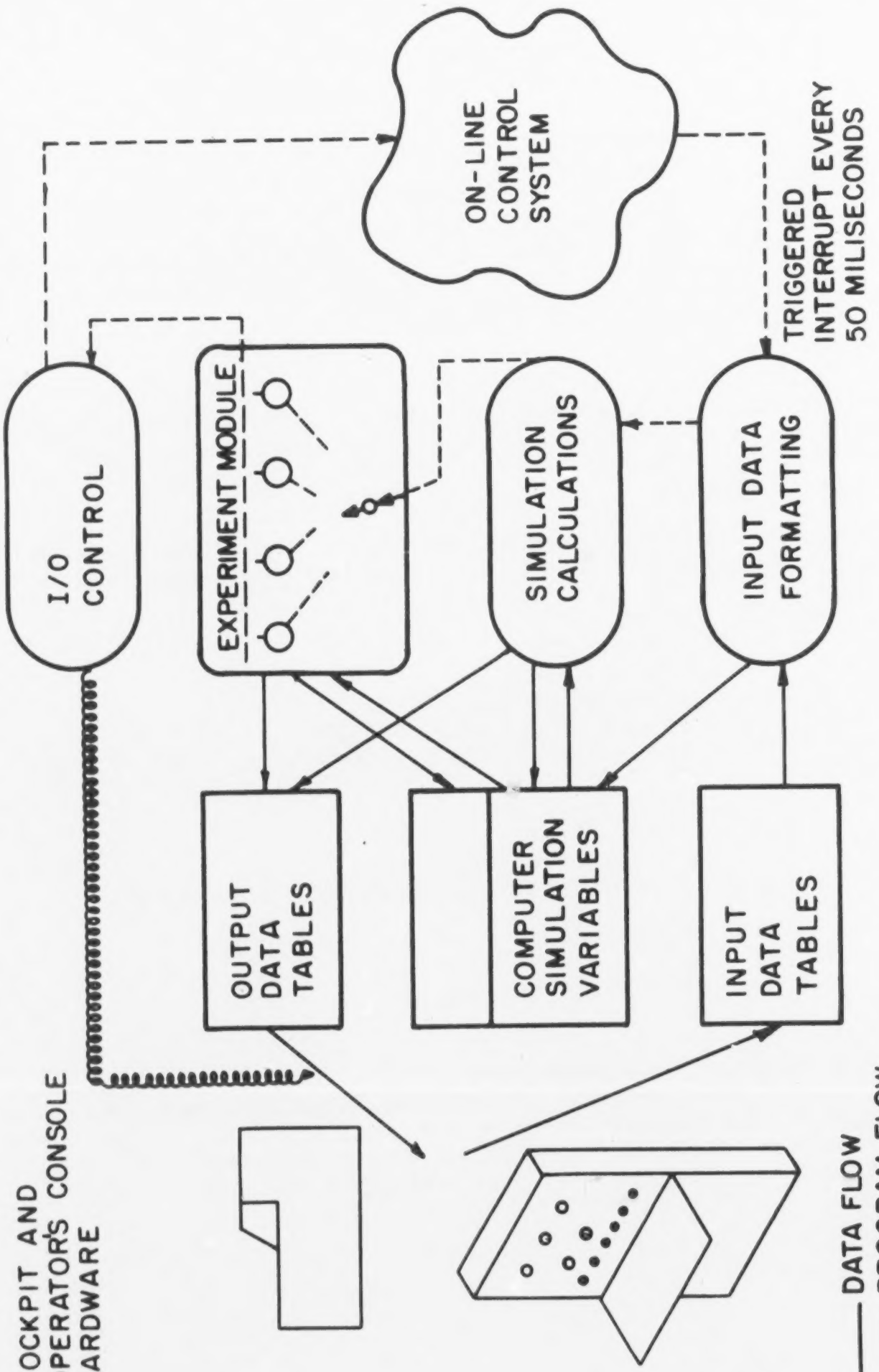

Figure 4. Experimental Module Partitioned Into States

```
START
NUMBER VAR1;
SET VAR1=ALT+MACH*5000;
IF ALT>20000 THEN HOLD;
IF -30<ROLL<30 THEN RECORD VAR1, ALT, PITCH;
IF FUEL<20000 THEN TYPE "LOW FUEL";
FINISH
```

The last statement poses a problem since when the fuel goes below 2000 lbs. the TYPE statement will be executed on each 50-millisecond simulation cycle, producing data faster than the teletype can absorb it.

The following reorganization of the program into distinct program states avoids the problem of continuous output.

```
START
NUMBER VAR1;
STATE EXPNORM;
SET VAR1=ALT+MACH*5000;
IF ALT>20000 THEN HOLD;
IF -30<ROLL< THEN RECORD VAR1, ALT, PITCH;
IF FUEL<2000 THEN NEXTSTATE TRANS;

STATE FILLUP;
SET FUEL=FUEL+10;
IF FUEL>10000 THEN NEXTSTATE EXPNORM;

STATE TRANS;
TYPE "LOW FUEL";
NEXTSTATE FILLUP;
FINISH
```

Now the actions have been partitioned into three sections for normal experiment actions, transition actions, and fuel replenishment. The normal actions are performed until the fuel gets low at which time the transition state (executed only once) causes a typeout. The program then executes the fillup state

every cycle until the test condition is met when the state is shifted back to experiment normal. Note that the sequence in which the states are entered is independent of the order in which they appear in the program text; logical flow is controlled exclusively by the NEXTSTATE command.

4.1.2.8 *Procedures*

In many cases it may be desirable to group a number of commonly used statements, name them, and then invoke their actions from a number of places in the program. Only the simplest of mechanisms is provided for this purpose; that of a procedure without parameters. Procedures can be declared and named within the declarations of a program as follows

```
PROC name;
   statement;
   statement;
      .
      .
      .
   statement;
ENDPROC
```

Later in the program, the procedure name can be used as a statement which invokes the execution of the group of statements in its definition. For example:

```
START
NUMBER INIT;

PROC LIMITS;
SET INIT=Ø;
IF 19950<ALT<20050 AND -2<ROLL<2 AND 0.7<MACH<0.8 THEN
  SET INIT=1;

IF INIT=1 THEN TYPE "SYSTEM INITIALIZED";
ENDPROC
```

```
STATE WARMUP;
LIMITS;
IF INIT=1 THEN NEXTSTATE GO;

STATE GO;
EVERY 2 RECORD ALT, ROLL, MACH;

FINISH
```

4.1.2.9 *Special Structures*

To facilitate the process of on-line data analysis and monitoring during an experimental run, special data forms and associated operations are provided.

On-Line Analysis. These special structures supplement the facilities for more complex off-line analysis of recorded data. The on-line features are restricted by the limited computation time available each cycle in excess of the simulation requirements. The user can declare and name a complex data structure of any of several special forms, cause data to be entered into it and output the results.

As a first example, mean and standard deviations of selected variables are often desired as a quick check on experiment performance. The user can declare and name a structure as follows:

```
MSD CHECK1 ALT, MACH, ETRIM, PITCH;
```

An MSD structure called CHECK1 is formed with four components, namely ALT, MACH, ETRIM, AND PITCH. In his program, the user can cause data to be entered into this structure with

```
UPDATE   CHECK1;
```

This update statement causes the current value of the variables mentioned to be accumulated properly in the structure. An output statement of the form

```
RECORD
PUNCH
LPRINT
TYPE      } CHECK1;
```

completes evaluation of the mean and standard deviation values for each of the four variables and causes an appropriately formatted text output. The values generated in this construct can only be accessed by outputting the structure as text.

Another structure which operates in a similar fashion is MINMAX which records the minimum and maximum variable values encountered during the updating of the structure. The necessary declaration is of the form

```
MINMAX CHECK2    ALT, MACH, ETRIM, PITCH;
```

The update and output methods are the same as those for a mean and standard deviation structure.

Histograms are a useful tool in analyzing experimental performance. The declaration of a histogram structure permits the specification of several histograms under one structure name. To define a histogram one must indicate the variable used and the appropriate "bin limits."

```
HISTO CHECK3
   ALT       1000, 3000, 7000, 12000,
   MACH      20 BINS FROM .3 TO 2.7,
   TEMP1     10 BINS FROM 1000 TO 1500;
```

The CHECK3 structure contains three histograms on the three variables with the indicated kinds of bin boundaries. The statement

```
UPDATE CHECK3;
```

will use the current values of the variables and update the counts in the appropriate bins. The structure CHECK3 can, for example, be output by

```
LPRINT CHECK3;
```

which will cause 3 histogram prints on the line printer.

For any of these special constructs, the statement

```
ZERO   structure name;
```

will initialize the structure contents. The statement

```
ZERO;
```

will initialize all of the special data structures declared in the program. Both the UPDATE and ZERO statements may specify a list of structures to be affected. For example,

```
ZERO   CHECK1, CHECK2;
UPDATE   CHECK1, CHECK3;
```

The basic concept common to these special constructs is that of associating specialized data entry and output routines with a behavioral science oriented data format. These complex facilities are invoked by simply declaring and naming a data structure of the desired type, indicating when to update the accumulation of data, and when and where to output the results. The necessary details are hidden within the BSPL system. Many possibilities for future expansion exist within this conceptual framework, for example, structures and routines for mean absolute error calculations or for multiple variable regression analysis.

Monitoring. Special constructs are also available for operating some of the TRADEC hardware used for monitoring the course of an experiment. Should the experimenter wish to obtain a time history of selected system variables on the strip chart recorder, he may declare a data structure similar to the following.

```
STRIPCHART TIMEHIST
ALT ON PEN1 FROM 4000 TO 8000
ROC ON PEN2 FROM -50.0 TO 50.0
MACH ON PEN3 FROM 0.5 TO 0.9;
```

For each recorder channel to be used, the name of the variable to be recorded and the desired range of values of the variable are specified.

The command

```
UPDATE TIMEHIST;
```

will cause the associated A/D converters to be appropriately updated according to the current values of the variables. Analog voltages will be maintained until another UPDATE command is executed.

The statement

```
ZERO TIMEHIST;
```

will cause all channels associated with this particular data structure to be zeroed. A pre-specified calibration routine is initiated by a statement of the form

```
CALIBRATE;
```

This routine will be similar to the one which is currently available on the TRADEC system.

4.2 On-Line Control System

4.2.1 General Characteristics

The on-line control system is used to control the operation of the F4 simulator, to initialize experimental runs, and to examine and modify the computer parameters associated with the simulation and the experiment. This system is *always active;* it is not bypassed when a simulated flight is in progress. The control system operates as a background job to the 50-millisecond simulation cycle, servicing each command as quickly as possible. Commands can be initiated from either the teletype or the card reader.

4.2.2 Commands

The commands available in the on-line control system are listed below. They are grouped into four categories: mode control, data control, state control, and I/O device control.

4.2.2.1 *Mode Control*

The operation of the F4 simulator is controlled by specifying the mode of the simulation. At any time the experiment operator can issue a command to force the simulator into one of its eight modes. These commands are:

FLY	(aircraft flying)
EXPERIMENT	(flying plus experiment module action)
HOLD	(flying suspended)
GROUND	(aircraft forced to ground)
DUMP	(record aircraft condition)
INITIALIZE	(restore aircraft conditions)
MAINTENANCE	(instrument test)
QUIT	(terminate experiment)

4.2.2.2 *Data Control*

The operator may initiate output on either the line printer, teletype, card punch, or magnetic tape by commands of the form

LPRINT
TYPE
PUNCH
RECORD } name, name, . . .name

which will create the appropriate text buffer and send it to the indicated device. The format of the output will be selected automatically by the BSPL to be compatible with the form of the data structure being output.

The values of numerical, text, and boolean variables can be changed by

SET name = value

as in the off-line language. In addition, a special shorthand command is provided which combines the functions of examining and modifying the values of numerical, text, and Boolean variables. The command is of the form

variable name: present value=new value

where the underlined portion of the statement is typed out by the on-line system. It is not necessary to specify a new value, and typeout of the present value may be suppressed. As examples:

ALT: 30000=20000
MACH: 1.05= (carriage return)
NAME= "SMITH"

The data type of each variable is known by the on-line system so the values of the variables are typed out in the correct format and new values can be checked to ensure that they are of the appropriate data type.

The special types of data structures for on-line data analysis such as histograms and mean and standard deviations can be initialized by the command

ZERO data structure name

The command

ZERO

will initialize all of these special data structures contained in the experiment specification program.

4.2.2.3 *State Control*

Since a BSPL program can be composed of several phases called states, a means of specifying the experiment state is needed.

The command

NEXTSTATE state name

changes the state of the experiment.

4.2.2.4 *I/O Device Control*

Several commands will change the input/output characteristics of the simulation, experiment, and on-line control system. The command

(break) or (EOM)	T	(teletype output)
	R	(magnetic tape)
	L	(line printer)
	P	(card punch)
	S	(strip chart recorder)
	C	(card reader)
	A	(all)

will stop the specified I/O device and clear its buffer. This feature can be used to suppress runaway outputs or to control the output of experiment debugging messages. The device will remain de-activated until the command

ACTIVATE { T R L P S C A }

turns on the device with a clear buffer. During program startup all devices are activated.

The console teletype and card reader can be used interchangeably as the source of commands, where the commands

CARD	(get commands from cards)
KEYBOARD	(get commands from teletype)
#	(get a variable a new value from teletype)
XXX	(do not print cards on teletype)
YYY	(yes, do print cards on teletype)

control this source. The keyboard is the initial command source.

4.2.3 Additional Features

When the on-line control system is waiting for a user command from the teletpye, an @ will be typed out. All commands are terminated by a carriage return, a semicolon, or the end of a card. The names of the commands have been chosen such that the first letter of the command is sufficient to identify it; however, the entire command name may be used. Hitting the rubout key on the teletype will erase the last letter of a partially typed command. Meaningful error messages will be typed out for invalid commands, undefined variable, data structure, and state names, and new values of the wrong data type. The special command

?

will list all the commands of the on-line control system on the teletype.

4.2.4 Initialization Example

The on-line control system rather than the stored experimental module is used to control the initialization of an experiment since this is inherently a background job. Typically, the experimenter will prepare a deck of cards to initialize an experimental run; the data values which vary with each experiment being input from the teletype. As an example, the experimenter types

```
CARD
```

to initiate the read-in of a card deck containing

```
HOLD
FUEL = 7000
SET NAME = #
DIFFICULTY = #
RECORD "EXPERIMENT 1", NAME, DIFFICULTY
EXPERIMENT
KEYBOARD
```

When the name of the pilot is requested, he types "SMITH" and when the level of difficulty is requested, he types 3. Then the name of the experiment and these values are recorded on magnetic tape, the experiment is started, and control is returned to the teletype.

SECTION V

IMPLEMENTATION OF THE BSPL

5.1 Implementation Tasks

The system design we have developed can be implemented in a series of relatively independent steps, each of which will provide an increased capability by itself. The implementation activities can be categorized in terms of the development of the following:

a. Basic system modifications
b. Background programs
c. New simulator modes
d. BSPL translator system

5.1.1 Basic System Modifications

The simulator program as it exists requires a certain amount of clean-up and reshaping to accommodate the concepts of the BSPL system. The mode and cycle control module must be extended to include the new modes which we have defined and must incorporate provisions for coupling in the object modules to be produced by the BSPL translator for each experiment. In addition, some minor rearrangement of the data input techniques will be required. Our design incorporates a somewhat different data recording scheme, replacing those programs that already exist for that purpose. In general, then, some effort will be needed to rearrange the control structure slightly and to modify the input and output techniques of the existing system. However, the basic organization of the real-time simulation program and the numerical modules which perform the actual flight calculations will remain unchanged.

5.1.2 Background Programs

The on-line control system will be implemented in the background, running in available time left over from each simulation cycle. The routines which perform the four basic on-line control system functions of mode control, data control, state control,

and I/O service control must be implemented appropriately in this background program. In addition, the system design incorporates expanded run-time output facilities, and these too must be included in the background. The routines to handle magnetic tape, line printer, card punch, etc., must be implemented, as well as the buffering and queueing schemes for data generated both in the simulation cycle and in the on-line control system.

5.1.3 New Simulator Modes

An important conceptual addition to the system is the ability to save an arbitrary aircraft state on tape, and later to restore those conditions to the simulation from the tape. We have suggested the addition of a dump mode and an initialize mode to the system to accomplish this feature. Implementing the dump functions required to save the aircraft state is relatively straightforward. However, restoring the saved conditions will require data input from tape, followed by appropriate actions in the simulation cycle to bring the aircraft smoothly to the desired state. Many of the cockpit and control console indicators are synchro-driven on a multi-turn basis and cannot be arbitrarily positioned. These indicators must be driven in a controlled way from a known position to some other desired position, and thus provide a complicating factor in restoring the aircraft cockpit to a previously saved state.

5.1.4 BSPL Translator System

The off-line language and translator are the means by which the experimenter creates additions to the basic real-time simulation system. The design we have developed contains a number of concepts which will be useful to an experimenter, e.g., experiment partitioning into phases called states, special data structures, etc. We believe that these concepts are important components of an experimenter's capability, and have presented them as part of an explicit language design.

Implementation issues for this language concern both the particular translator implementation and the computer compatability problems of loading the resulting object module with the object code of the basic simulator system. To allow the on-line control system to access experiment and simulator parameters symbolically, the symbol table generated in the translator must be included in the run-time computer data.

5.2 Implementation Strategy

An extensive amount of effort is likely to be required for the complete implementation of the proposed BSPL. We therefore recommend that implementation proceed in stages so that benefits to the NTDC research program may be realized as rapidly as possible. It may well be that the limited resources available for implementation will dictate utilizing an existing translator to produce initial experimental modules. For example, FORTRAN with a suitable set of subroutines could provide much of the capability needed. Alternately, a set of procedures (macros) could be defined for the Meta-Symbol assembler to provide the off-line translator capability. Although this approach would involve a relatively clumsy syntax, a significant improvement to current operating procedures would be realized.

Several stages of sophistication in translator development are possible, and initial attention should be focused not on a fancy language and translator but on the system problems of coupling and linking to the basic simulator programs. With these resolved, the development effort remaining can be devoted to a special BSPL translator for the language formally defined here (Appendix A). Ultimately, after some real experience, the language may be redesigned and reimplemented more closely to meet the needs of the experimenters using it.

SECTION VI

REFERENCES

1. Kapsis, P.A., et.al., "Software Documentation for the Research Tool Digital Computer System, Volume I Math Model Report," Technical Report NAVTRADEVCEN 67-C-0196-7, Naval Training Device Center, Orlando, Florida, June 1968 (Revised September 1969).

2. Erickson, Eleanor S., Kapsis, Patricia B., and Ciolkosz, M. Denise, "Software Documentation for the Research Tool Digital Computer System, Volume II Program Report," Technical Report NAVTRADEVCEN 67-C-0196-7, Naval Training Device Center, Orlando, Florida, January 1969 (Revised September 1969).

3. Lowes, A.L., et.al., "Improving Piloting Skills in Turbulent Air Using a Self-Adaptive Technique for a Digital Operational Flight Trainer," Technical Report NAVTRADEVCEN 67-C-0034-2, Naval Training Device Center, Orlando, Florida, August 1968.

APPENDIX A

FORMAL SPECIFICATION OF THE OFF-LINE LANGUAGE

[program]:: = START [declarations][state definition]$_1^\infty$ FINISH

[declarations]:: = [variable declaration]$_0^\infty$ [special structure declaration]$_0^\infty$ [procedure declaration]$_0^\infty$

[variable declaration]:: = NUMBER [variable list];| BOOLEAN [variable list]; | TEXT [variable list];

[special structure declaration]:: = MSD [structure name] [variable list]; | MINMAX [structure name] [variable list];| HISTO [structure name] [[variable name] [integer] BINS FROM [number] TO [number] | [variable name] [[number],]$_1^\infty$]$_1^\infty$; | STRIPCHART [structure name] [[variable name] ON [pen designator] FROM [number] TO [number],]$_1^\infty$;

[procedure declaration]:: = PROC [procedure name]; [statement]$_1^\infty$ ENDPROC

[state definition]:: = STATE [state name]; [statement]$_1^\infty$

[statement]:: = SET [assignment list]; | RECORD [output list]; |
LPRINT [output list]; | TYPE [output list]; | PUNCH [output
list];| HOLD; | INITIALIZE; | FLY; | GROUND; | DUMP; | QUIT; |
IF [boolean expression] THEN [statement] |
IF [boolean expression] THEN [statement] ELSE [statement] |
EVERY [integer] [statement] | ONCE [statement] |
[procedure name]; | NEXTSTATE [state name]; |
UPDATE [structure list]; | ZERO [structure list]; | ZERO; |
CALIBRATE; | <[statement]$_2^{\infty}$>

[variable list]:: = [[variable name],]$_1^{\infty}$ | [variable name]
[, [variable name]]$_0^{\infty}$

[structure list]:: = [[structure name],]$_1^{\infty}$ |[structure name]
[, [structure name]]$_0^{\infty}$

[output list]:: = [[output element],]$_1^{\infty}$ | [output element][,
[output element]]$_0^{\infty}$

[output element]:: = [variable name] | [structure name] |"[text]"

[assignment list]:: = [[variable name] = [arithmetic expression],]$_1^{\infty}$

| [variable name] = [arithmetic expression] [, [variable name]
= [arithmetic expression]]$_0^{\infty}$

[arithmetic expression]:: = [variable name] | [number] |
[unary operator][arithmetic expression] | [arithmetic
expression][binary operator][arithmetic expression] |
([arithmetic expression])

[unary operator]:: = + | - | ABSV | SQRT

[binary operator]:: = + | - | * | /

[boolean expression]:: = [arithmetic expression][relational operator]
[arithmetic expression] | [arithmetic expression]
[relational operator][arithmetic expression][relational operator]
[arithmetic expression] | [boolean expression] AND
[boolean expression] | [boolean expression] OR [boolean
expression] | NOT [boolean expression] | ([boolean expression])

[relational operator]:: = > | < | = | <= | >= | ¬ =

A.1 Command List for On-Line Control System

1. Mode Control

FLY
EXPERIMENT
HOLD
GROUND
DUMP
INITIALIZE
MAINTENANCE
QUIT

2. Data Output

LPRINT
TYPE
PUNCH
RECORD

} name, name, ... name

3. Data Input

SET name = value
variable name: present value = new value
ZERO data structure name
ZERO
NEXTSTATE state name

4. I/O Device Control

break
ACTIVATE

{ T
R
L
P
S
C
A

CARD
KEYBOARD
#
XXX
YYY

5. Help

?

APPENDIX B

A BSPL PROGRAM TO CONDUCT AN INPUT-ADAPTIVE EXPERIMENT

This appendix contains a sample program written in the proposed BSPL. The program is designed to allow the experimenter to conduct the input-adaptive experiment that was referenced as the "prototype experiment" in Section III (Ref. 3). This experiment is familiar to some of the personnel at NTDC and, we believe, is representative of experiments that will be performed in the near future. Most importantly, this example serves as a suitable vehicle to demonstrate the various features of the BSPL.

This program is intended to be executable on the TRADEC system once the BSPL has been implemented. It may thus serve as a means for debugging and demonstrating the language. Accordingly, the BSPL program has been written using variable names as they currently exist in the TRADEC system. The quantities represented by these variables are defined in a glossary at the end of this section. Additional variables are declared at the beginning of the experimental module.

The program would be implemented as three physical entities: a deck of cards to facilitate on-line control of the daily initialization procedure, another deck of cards to facilitate run-to-run initialization, and the stored experimental module to simulate the adaptive routine and to perform recording, monitoring, and analysis operations. These programming units are listed in the following three sections of this appendix. Each listing is preceded by commentary to point out the important features that are demonstrated by the program.

B.1 Session Initialization

B.1.1 Commentary

The following program would be written on cards to facilitate the initialization that is performed at the beginning of THE EXPERIMENTAL SESSION. This routine first places the simulation in the "ground" state, initializes the experimental run

number to one, and allows the experimenter to specify various program constants related primarily to the input-adaptive routine. In order to execute this program, the user types "CARD" on his teletype. Control is then exercised by the card reader, which reads each card and performs the required operation. The last card contains the statement "KEYBOARD" which returns control to the user's teletype.

The "set" command is used extensively in the initialization routines to cause the computer to "ask questions" of the experimenter. For example, the statement

```
SET TITLE=#
```

which is contained on the third card will cause the following message to appear on the teletype:

```
TITLE=
```

The program then pauses and awaits input from the operator via his teletype. Once the user has entered a legitimate input string, the statement contained on the next card is executed.

B.1.2 Program Text

```
XXX
GROUND
SET TITLE=#
SET RUN=1
TYPE "DESIRED ALTITUDE, MINIMUM AND MAXIMUM TOLERANCES:"
SET DALT=#
SET MINALT=#
SET MAXALT=#
TYPE "ERROR CRITERION FOR ADAPTIVE SCHEME:"
SET ERRCRIT=#
TYPE "MAX ALLOWABLE ROUGH AIR MULTIPLIER:"
SET CLRA=#
TYPE "INSERT DECK FOR RUN INITIALIZATION, TYPE 'CARD' TO CONTINUE."
KEYBOARD
```

B.2 Run Initialization

B.2.1 Commentary

This program, also stored on cards, facilitates the initialization that is performed at the beginning of each experimental run. The stored statements cause the typeout of certain information to identify the experiment, allow the experimenter to quantify various experimental parameters, and then return control to the keyboard for corrections. At this stage, the experimenter can change the identification information or re-specify one or more experimental variables. To do so, he would use the on-line shorthand command facility described in Section 4.2.2.2. When all corrections have been made, the experimenter again types "CARD" to continue operation of the stored initialization program. Identification information is recorded on the data tape, the simulator is initialized, and all special data structures are initialized. The initial state of the stored experimental module is specified, the system is placed in the "experiment" mode to start the experiment, and control of the on-line system is returned to the keyboard.

B.2.2 Program Text

```
XXX
TYPE TITLE
TYPE RUN
SET SUBJECT=#
TYPE "ADAPTIVE RATE CONSTANT:"
SET KADAPT=#
TYPE "INITIAL ROUGH AIR MULTIPLIER:"
SET CLRA=#
TYPE "MAKE CORRECTIONS", "MOUNT DATA TAPE ON UNIT 1", "WHEN
     FINISHED, TYPE 'CARD' TO CONTINUE."
```

```
KEYBOARD
RECORD TITLE, RUN, SUBJECT, KADAPT, CLRA
INITIALIZE
ZERO
TYPE "TYPE 'C' WHEN READY FOR PILOT CONTROL OF SIMULATOR"
NEXTSTATE ALTINIT
EXPERIMENT
KEYBOARD
```

B.3 Stored Experimental Module

B.3.1 Commentary

This portion of the user program would be stored initially in symbolic form on cards, converted to machine code by the BSPL translator, and loaded into the SIGMA 7 with the basic simulation code. The experimental module would be activated each simulation cycle whenever the system was in the "experiment" mode.

The example that follows makes use of the capability to create special procedures and special data structures. Two analysis structures are declared (mean and standard deviation, histogram) along with the special structure for operating the strip chart recorder. Two user-created procedures are defined. The procedure named "adapt" modifies the simulation cycle to perform the desired adaption of the rough-air input. The adaptive equations given in Section 3.2.1 of this report are simulated, and a check is performed to keep the rough-air amplitude within specified limits (as was done by Lowes *et al.* in Ref. 3). A second procedure, "check," is defined to indicate when the aircraft is flying straight and level at the desired altitude.

The program proper consists of six program states. The first state simply performs a test to determine whether or not the airplane is in the proper initialization configuration. This configuration may be different from that which was input to the simulator at the beginning of the flight. (For example, the experimenter may wish the flight to proceed at a new altitude.) When the pilot has attained the desired straight-and-level conditions, control passes to the state named "pause," which places the simulation in HOLD, and types a message on the teletype. The experimenter now has the option to save the new initial condition on magtape for future restoration.

The operator types "experiment" when he is ready for the flight to continue. The flight then proceeds with the experiment module in the "warmup" state for 30 seconds, so that the pilot may become used to the input-adaptive routine which is now activated for the first time. Recording on the strip chart is also started during this phase of the flight.

The bulk of the flight is performed in the "flight" state. Simulation of the input-adaptive equations continues during this period, performance measures are collected, and data are recorded on magtape. Data recording is performed every other simulation cycle; all other operations are performed each cycle. At the end of the experimental period, the pilot restores the aircraft to its initial condition. When this configuration is attained, the "output" state of the experiment module causes the performance measures to be calculated and printed, types a message to the operator, and invokes the "quit" command. In order to perform another experimental run, the operator must next load the cards for the run initialization into the card reader and execute the initialization routine.

B.3.2 Program Text (with annotations)

```
START
NUMBER RUN, TIME, INPUT, ERROR, ERRCRIT,
  MAXCLRA, MINALT, MAXALT, KADAPT;
TEXT TITLE;

STRIPCHART TIMEHIST
ALT ON PEN1 FROM MINALT TO MAXALT,
RC ON PEN1 FROM -50 TO 50,
MACH ON PEN3 FROM 0.5 TO 0.9;

MSD SCORES ALT, RC, INPUT;

HISTOGRAM RUFFAIR
CLRA 10 BINS FROM 0 TO 1.0;


PROC ADAPT;
SET INPUT=DRA*CLRA, TIME=TIME + 0.05;
SET ERROR=ERRCRIT-ABSV(ALT-DALT);
SET ERRSQR=ERROR*ERROR;
IF ERROR>Ø THEN SET CLRA=CLRA + KADAPT*ERRSQR
  ELSE SET CLRA=CLRA-KADAPT*ERRSQR;
IF CLRA>MAXCLRA THEN SET CLRA=MAXCLRA ELSE
  IF CLRA<Ø.Ø THEN SET CLRA=Ø.Ø;
ENDPROC
```

declaration of variables

declaration of special data structures

procedure to perform input adaptation

```
PROC CHECK;
IF MINALT<ALT<MAXALT AND -2<RC<2 AND -0.2<QA<0.2
  THEN SET LEVEL=1 ELSE SET LEVEL=0;
ENDPROC
```

procedure to check aircraft configuration

```
STATE ALTINIT;
CHECK;
ONCE SET TIME=Ø.Ø;
IF LEVEL=1 THEN NEXTSTATE PAUSE;
```

first program state; check aircraft configuration

```
STATE PAUSE;
TYPE "DESIRED ALTITUDE ATTAINED.",
 "DUMP ON TAPE IF DESIRED",
"TYPE 'NEXTSTATE WARMUP' TO CONTINUE EXPERIMENT.";
HOLD;
```

operator to sense initial configuration

```
STATE WARMUP;
UPDATE TIMEHIST;
ADAPT;
IF TIME=30.0 THEN NEXTSTATE FLIGHT;
```

provide pilot with 30-second warmup

```
STATE FLIGHT;
ADAPT;
UPDATE TIMEHIST,SCORES,RUFFAIR;
EVERY 2 RECORD ALT,RC,PA,QA,RA,DSS,CLRA,INPUT;
IF TIME=270.0 THEN NEXTSTATE TERM;
```

actual experiment

```
STATE TERM;
CHECK;
IF LEVEL=1 THEN NEXTSTATE OUTPUT;
```

check final aircraft configuration

```
STATE OUTPUT;
SET RUN=RUN+1;
LPRINT SCORES,RUFFAIR;
TYPE "END OF FLIGHT.";
QUIT;

FINISH
```

} perform immediate post-flight operation

B.4 Glossary of Variable Names

The following variable names do not have to be declared in the experimental module since they are defined within the basic simulation code. The quantities represented by these names are given below.

Variable Name	Quantity Represented
ALT	Altitude
DALT	Desired Altitude
MACH	Aircraft Mach Number
RC	Rate of Climb
CLRA	Multiplier on Rough Air Intensity
DRA	Rough Air Amplitude
CLOCK	Cycle Count
PA	Roll Rate
QA	Pitch Rate
RA	Turn Rate
DSS	Stabilator Stick Deflection

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE

DEVELOPMENT OF BEHAVIORAL SCIENCE PROGRAMMING LANGUAGE

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Final Report covering period August 1969 to June 1970

5. AUTHOR(S) *(First name, middle initial, last name)*

Sheldon Baron, William H. Levison, Raymond S. Nickerson,
William R. Sutherland, Elaine L. Thomas

6. REPORT DATE
February 1971

7a. TOTAL NO. OF PAGES
54

7b. NO. OF REFS
3

8a. CONTRACT OR GRANT NO.
N61339-70-C-0046

b. PROJECT NO.
8504-1A

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
BBN Report No. 1979

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
NAVTRADEVCEN 70-C-0046-1

10. DISTRIBUTION STATEMENT

Approved for public release; distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
U.S. Naval Training Device Center
Orlando, Florida 32813

13. ABSTRACT Design criteria and an implementation plan are specified for a Behavioral Science Programming Language (BSPL) to be used for programming experiments on the computer-driven, high-performance aircraft simulation facility at NTDC. The BSPL was conceptualized with the following goals in mind: (a) the language should take full advantage of the existing hardware and software facilities; (b) it should be rich enough in structure to allow the programming of meaningful experiments, yet simple enough to be readily learned by the non-programmer; (c) it should be usable by individuals who do not have intimate knowledge of the simulation program; (d) it should be incrementally implementable; and (e) it should be capable of graceful expansion and elaboration.

The language requirements are determined in large part by the nature of the existing facility on the one hand, and the types of experiments that are contemplated on the other. This report begins, therefore, with a description of the present facility, a conceptualization of a prototype experiment, and the implication of these factors for the design of the BSPL. A proposed design is then presented along with an implementation plan. The implementation plan partitions the development of the language in such a way that incremental improvements in the facility can be realized as the plan is being carried out. A formal description of the language and a sample program are included as appendices.

DD FORM 1 NOV 65 1473

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Behavioral Science						
Programming Language						
Adaptive Training						
Computer-aided Experimentation						